RENDERING A USER INTERFACE

FIELD OF THE INVENTION

The present invention relates to rendering user interfaces and in particular to rendering user interfaces for communications devices.

BACKGROUND OF THE INVENTION

Communications devices, such as, for example mobile telephones and PDAs incorporate display screens of increasing size and resolution. Given the limitations in processing power of these devices it is desirable to provide users with an attractive user interface that facilitates the use of the device and provides a fast response to user inputs . For some devices, such as mobile telephones, there is significant interest in providing user interfaces that can be readily and easily updated by the user and/or the network operator so that content for updating user interfaces can be deployed to users. Known approaches tend to either lack the required flexibility or require significant and undesirable levels of processing power.

SUMMARY OF THE INVENTION

According to a first aspect of the present invention there is provided a method of rendering a user interface for a device, the method comprising the steps of providing a plurality of actors, each of the plurality of actors being associated with a user interface element and comprising one or more attributes defining the respective actor; providing a

renderer to receive one or more attributes from one or more of the plurality of actors and rendering the user interface in accordance with the received attributes.

According to a second aspect of the present invention there is provided a data carrier comprising computer executable code for performing the above-described method.

According to a third aspect of the present invention there is provided a device comprising: a user interface, the user interface comprising one or more user interface elements; a plurality of actors, each of the plurality of actors being associated with a user interface element and comprising one or more attributes; and a renderer, the renderer being configured, in use, to interpret the attributes associated with one or more of the plurality of actors and to render the user interface accordingly.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a schematic depiction of a system incorporating the present invention;

Figure 2 depicts in greater detail the structure and operation of server 100;

Figure 3 shows a schematic depiction of the software 400 for the mobile devices 300;

Figure 4 shows a schematic depiction of the content toolset 200; and

Figure 5 shows a schematic depiction of a device that comprises a user interface according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The invention will now be described by way of illustration only and with respect to the accompanying drawings, in which Figure 1 shows a schematic depiction of a system comprising server 100, content toolset 200, mobile devices 300, operational support systems (OSSs) 700, content feeds 500 and user interface (UI) sources 600. In use, the server 100 communicates content data and UI data to the mobile devices 300, 301, ..., each of which comprise software package 400. The server 100 interfaces with OSSs 700, with the OSSs being those conventionally used to operate mobile networks, for example billing, account management, etc. The server 100 further interfaces with the content toolset 200: the content toolset receives data from UI sources 600, 601, ..., and packages the UI data such that the server can transmit the packaged UI data to the software packages 400 comprised within the mobile devices 300. The server receives data from a plurality of content feeds, and this data is processed and packaged such that it can be sent to the software packages 400 or so that the mobile devices 300 can access the data using the software package 400.

The system can be envisaged as being divided into three separate domains: operator domain 50 comprises the systems and equipment operated by the mobile network operator (MNO); user domain 60 comprises a plurality of mobile devices and third-party domain 70 comprises the content feeds and UI feeds that may be controlled or operated by a number of different entities.

Figure 2 depicts in greater detail the structure and

operation of server 100. Server 100 comprises publishing component 110 and content server component 150. Publishing component comprises database 111, import queue 112, content toolset interface 113, user interface 114 & catalogue 115. In operation, the publishing component receives content from the content toolset at the content toolset interface. The content is presented in the form of a parcel 210a, 210b, etc,(see below) comprising one or more Trigs and one or more Triglets. A trig is a user interface for a mobile device, such as a mobile telephone and a triglet is a data file that can be used to extend or alter a trig. If a parcel comprises more than one trig then one of the Trigs may be a master trig from which the other Trigs are derived.

Figure 3 shows a schematic depiction of the software 400 for the mobile devices 300, which comprises a mark-up language renderer 410, update manager 420, network communication agent 425, resource manager 430, virtual file system 435, actor manager 440, a plurality of actors 445a, 445, ..., native UI renderer 450, support manager 460, trig manager 465 and mark-up language parser 470.

The software may operate using TrigML, which is an XML application and that mark-up language renderer 410 renders the TrigXML code for display on the mobile device 300. The mark-up language renderer also uses the TrigML Parser to parse TrigML resources, display content on the device screen and controlling the replacement and viewing of content on the handset. The native UI renderer is used to display UI components that can be displayed without the use of TrigML, and for displaying error messages.

The software 400 is provisioned and installed in a device specific manner. Similarly, software upgrades are handled in a device specific manner. The software may be provisioned in a more limited format, as a self-contained application that renders its built in content only: i.e. the software is provisioned with a built-in trig but additional trigs cannot be added later. The supplied trig may be upgraded over the air.

The trig manager 465 presents an interface to the resource manager 430 and the mark-up language renderer. It is responsible for trig management in general. This includes: persisting knowledge of the trig in use, changing the current trig, selection of a trig on start-up, selection of a further trig as a fall back for a corrupt trig, maintaining the set of installed trigs, identifying where a particular trig is installed to the resource manager and reading the update channel definitions of a trig and configuring the update manager appropriately.

The resource manager provides an abstraction of the persistent store on device, i.e. storing the files as real files, or as records in a database. The resource manager presents a file system interface to the mark-up language renderer and the update manager. It is responsible for handling file path logic, distinguishing between real resource files and actor attributes, mapping trig-relative paths onto absolute paths, interfacing with the trig manager and providing a modification interface to the update manager.

The Resource Manager is also responsible for ensuring the integrity of the resources stored in the persistent store,

especially in the face of unpredictable interruptions such as loss of device power. The Resource Manager has no knowledge of the trig currently used. Its interface is thread safe (as it may be used by both the Update Manager and the Renderer from different threads.

The Update Manager handles the reception and application of Trigs and Triglets. The Update Manager presents an interface to the Renderer and the trig Manager and is responsible for: the initiation of manual updates when instructed to by the Renderer; controlling and implementing the automatic update channel when so configured by the trig manager; indicating the progress of a manual update and recovering an Update following unexpected loss of network connection and/or device power. The update packet format may be defined as a binary serialisation of an XML schema.

The Support Manager provides an interface for other components to report the occurrence of an event or error. Depending on the severity of the error, the Support Manager will log the event and/or put up an error message popup

XML is a convenient data formatting language that is used to define the update packet format as well as TrigML content. For bandwidth and storage efficiency reasons, text XML is serialised into a binary representation. Both update packets and TrigML fragments are parsed by the same component, the mark-up language parser. Any further use of XML in the software must use the binary XML encoding and therefore re-use the parser.

The Actor Manager 440 looks after the set of actors 445

present in the software. It is used by: the renderer when content is sending events to an actor; actors that want to notify that an attribute value has changed and actors that want to emit an event (see below).

The software may comprises a multi-threaded application running a minimum of two threads, with more possible depending on how many and what sort of actors are included. The software runs mostly in one thread, referred to as the main thread. The main thread is used to run the renderer which communicates synchronously with other components. Actors always have a synchronous interface to the Renderer. If an actor requires additional threads for its functionality, then it is the responsibility of the Actor to manage the inter-thread communication. A light messaging framework may be used to avoid unnecessary code duplication where many actors require inter-thread communication. It will be understood that it is also possible to implement the software using a single threaded operation.

In addition to the main thread, the update manager runs a network thread. The network thread is used to download update packets and is separate from the main thread to allow the renderer to continue unaffected until the packet has arrived. The Update Manager is responsible for handling inter-thread messaging such that the Update Manager communicates synchronously with the Renderer and Resource Manager when applying the changes defined in an Update Packet.

The memory allocation strategy of the software is platform specific. On MIDP platforms, the software simply uses the

system heap and garbage collector for all its memory requirements. Garbage collection is forced whenever a content replacement event occurs in an attempt to keep the garbage collection predictable and not suffer unexpected pauses in operation. It is assumed that any memory allocation might fail, in which case the software will delete all its references to objects, garbage collect, and restart - assuming that the software has already successfully started up and rendered the first page.

On C++-based platforms, a mixture of pre-allocation and on-demand allocation will be made from the system heap. All memory required for start-up is allocated on-demand during start-up, with any failures here causing the exit (with message if possible) of the software. Following successful start-up, memory needed for rendering the content document model is pre-allocated. Provided content is authored to use less than a defined limit, it is guaranteed to render. Additional use is made of RAM for various caches needed for fast operation of the software. Where memory conditions are low, these caches will be released resulting in slow rendering performance from the software.

Errors that are severe enough to disrupt the normal operation of the software must result in a pop-up dialog box. The dialog box contains one of a small number of internationalised error messages (internationalised versions of these strings may be compiled into the software at build-time with the version of an error string to display being determined by the relevant language setting on the device). To keep the number of messages to a minimum, only a few generic problems are covered.

To allow for support situations, error dialogs also display an error code as a 4-digit (16-bit) hex string. Each error code is associated with a description text that can be used by support staff to determine the nature of a problem with the software. Errors that occur in the software and that are not severe enough to halt its operation may be logged by the Support Manager component. The Support Manager can be queried by the user typing special key sequences. The Support Manager can also supply its error log to a server via an HTTP GET or POST method.

The Renderer receives information regarding the key press. If there is no behaviour configured at build time for a key, it is sent as a TrigML content event to the current focus element. The content event is then handled as defined by TrigML's normal event processing logic.

For example, if a key is pressed down, a 'keypress' event is delivered to the Renderer with a parameter set to they relevant key. When the key is released, a '!keypress' event is delivered to the Renderer. If a key is held down for a extended period of time, a 'longkeypress' event is delivered to the renderer. On release, both a '!longkeypress' and a '!keypress' event are delivered to the Renderer.

Whenever the software is started, it executes the following actions:

- Check for, and continue with, interrupted Update processing;
- Check for, and process, Updates residing in the file system (either pre-provisioned, or installed to the file

system by some other means);

- If known, start the current trig (which may be the last run trig);
- If a current trig is not set, a trig that has been flagged as a 'default' trig can be started.
- Failing the presence of a default trig, the first valid trig by alphabetical order of name will be selected.

A trig is started by loading the defined resource name, start-up/default. The TrigML defined in start-up/default is parsed as the new contents for the content root node.

The first time a trig is run by the software following its installation, the trig is started by loading the resource name startup/firsttime. The software may record whether a trig has been run or not in a file located in the top level folder for that trig. Dependent on the platform used by the mobile device, the automatic start-up of the software may be set as a build-time configuration option. Furthermore, placing the software in the background following an auto-start may also be a build-time configuration option.

A launcher may appear to the user as an application icon and selecting it starts the software with a trig specified by that launcher (this trig may be indicated by a launcher icon and/or name). When using a launcher to start a trig, it is possible to specify an 'entry point' parameter. The parameter is a resource name of a file found in the 'start-up' folder. This file is not used if the trig has never been run before, in which case the file called 'firsttime' is used instead.

The software uses content resource files stored in a virtual

file system on the device. The file system is described as virtual as it may not be implemented as a classical file-system, however, all references to resources are file paths as if stored in a hierarchical system of folders and files.

Details regarding the arrangement of the file-system for an embodiment of the present invention are given below in Appendix A. Furthermore, the software stores some or all of the following information: usage statistics; active user counts; TrigManager state; TrigML fragments & update channel definition (serialised as binary XML); PNG images; plain text, encoded as UTF-8 OTA and then stored in a platform specific encoding; other platform specific resources, e.g. ring tone files, background images, etc.

Files in the file system can be changed, either when an actor attribute value changes, or when a file is replaced by a triglet. When files in the /attrs directory change, the Renderer is immediately notified and the relevant branches of the content tree are updated and refreshed. When images and text resources are changed, the Renderer behaves as if the affected resources are immediately reloaded (either the whole content tree or just the affected branches may be refreshed). When TrigML fragments are changed, the Renderer behaves as if it is not notified and continues to display its current, possibly out of date, content. This is to avoid the software needing to persist <include> elements and the <load> history of the current content.

The software 400 is provisioned to mobile devices in a device specific method. One or more Trigs can be provisioned as part of the installation, for example, stored as an

uncompressed update packet. On start-up, the packet can be expanded and installed to the file-system.

The Actors 445 are components that publish attribute values and handle and emit events. Actors communicate with the Renderer synchronously. If an actor needs asynchronous behaviour, then it is the responsibility of the actor to manage and communicate with a thread external to the main thread of the Renderer.

Actor attributes may be read as file references. Attributes are one of four types: a single simple value; a vector of simple values; a single structure of fields, each field having a simple value; or a vector of structures. Attributes may be referenced by an expression using an object.member notation similar to many object-orientated programming languages:

```
<image res="signallevels/{protocol.signalstrength}"/>
```

When needed as a file, an attribute is accessed via the /attrs folder.

```
<text res="/attr/network/name">
```

An Actor can be messaged by sending it an event with the <throw> element. Events emitted by actors can be delivered to the content tree as content events: these can be targeted at an element Id or 'top'. The interface to an actor is defined by an Actor Interface Definition file. This is an XML document that defines the attributes, types, fieldnames, events-in and parameters, and events out. The set of actors

is configurable at build-time for the software. Appendix B gives an exemplary listing of some actors that may be used, along with the associated functions or variables.

Updates comprise a new trig (a new or replacement UI) or a triglet (a modification to an existing trig) and may be regarded as modifications to the software file-system. The Update Manager to determine what needs changing in the file-system by reading a packet. Update Packets may be downloaded over the air by the software 400 using HTTP, or other suitable transport mechanisms, wrapped in a device-specific package format or pre-provisioned with the installation of the software itself.

Updates may be triggered by a number of means, which include

- the software checking for interrupted Update processing on start-up
- the software checking for pre-installed Update Packets on start-up
- automatically as configured by an Update Channel
- user initiation
- the device receiving a special SMS

In order to successfully render the user interface of a mobile device, the mark-up language must have the following qualities: concise page definitions, consistent layout rules, be implementable in a compact renderer, provide multiple layering and arbitrary overlapping content, event model, require the repaint of only the areas of the display that have to change between pages of the UI, include hooks to the platform for reading property values receiving events and sending events, extensible, and be graphically flexible.

TrigML provides these features and Appendix C gives an overview of the elements and attributes that provide the desired functionality.

It is desirable that the cost of re-branding UIs and producing a continual stream of updates is minimal. This is enabled by providing an efficient flow of information from the creative process through to the transmission of data to users.

A container, referred to as a parcel, is used for UIs, UI updates, and templates for 3rd party involvement. Parcels contain all the information necessary for a 3rd party to produce, test and deliver branded UIs and updates. Figure 4 shows a schematic depiction of the content toolset 200, which comprises scripting environment 220, test and simulation environment 230 and maintenance environment 240

The parcel process comprise five processing stages:

1) The scripting environment 220 provides the means to design the template for one or more UIs and the update strategy for UIs based on that template.

2) The maintenance environment 240 provides for rapid UI and update production in a well-controlled and guided environment that can be outsourced to content providers.

3) The maintenance environment 240 'pre-flight' functionality allows the deployment administrator to check and tune the UIs and updates that they receive from 3rd parties.

4) The publishing component 110 provides management of UIs and updates at the deployment point, including the staging of new releases.

5) The publishing component 110 enables the automatic

generation of updates from live content feeds.

Many different UIs can be derived from a common base. Typically the common base would implement most of the interface itself, and Trigs derived from it would implement small variations on it, such as branding. A Triglet can be derived from a Trig, and it can override any of the resources from the parent Trig that it chooses to (optionally it may introduce its own resources). Note that "resources" here also refers to TrigML, so the behaviour and layout of a Trig can be modified by a Triglet just as easily as it replacing a single image or piece of text.

A Parcel may comprise one or more base Trigs (i.e. a Trig ~~that is not derived from any other trig), one or more~~ multiple Trigs derived from a base Trig, a plurality of triglets derived from any of the trigs and a plurality of triglets derived from other triglets.

Figure 5 shows a schematic depiction of a device 800 that comprises a user interface according to an embodiment of the present invention. The device comprises a display 810 that displays the user interface 815 and user interface means 820, that enable the user to interact with the user interface 815. A processor 830 executes the software that is stored within one or more storage means 840 and there may be provided one or more wireless communication interfaces 850, to enable communication with other devices and/or communication networks. One or more batteries 860 may be received to power the device, which may also comprise interfaces to receive electrical power and/or communication cables.

The nature of these components and interfaces will depend upon the nature of the device. It will be understood that such a user interface can be implemented within a mobile or cellular telephone handset, but it is also applicable to other portable devices such as digital cameras, personal digital organisers, digital music players, GPS navigators, portable gaming consoles, etc. Furthermore, it is also applicable to other devices that comprise a user interface, such as laptop or desktop computers.

The user interface means may comprise a plurality of buttons, such as a numerical or alpha-numerical keyboard, or a touch screen or similar. One or more storage devices may comprise a form of non-volatile memory, such as a memory card, so that the stored data is not lost if power is lost. ROM storage means may be provided to store data which does not need updating or changing. Some RAM may be provided for temporary storage as the faster response times support the caching of frequently accessed data. The device may also accept user removable memory cards and optionally hard disk drives may be used as a storage means. The storage means used will be determined by balancing the different requirements of device size, power consumption, the volume of storage required, etc.

Such a device may be implemented in conjunction with virtually any wireless communications network, for example second generation digital mobile telephone networks (i.e. GSM, D-AMPS), so-called 2.5G networks (i.e. GPRS, HSCSD, EDGE), third generation WCDMA or CDMA-2000 networks and improvements to and derivatives of these and similar networks. Within buildings and campuses other technologies such as Bluetooth, IrDa or wireless LANs (whether based on

radio or optical systems) may also be used. USB and/or FireWire connectivity may be supplied for data synchronisation with other devices and/or for battery charging.

Computer software for implementing the methods and/or for configuring a device as described above may be provided on data carriers such as floppy disks, CD-ROMS. DVDs, non-volatile memory cards, etc.

This application claims the benefit of UK Patent Application number 0403709.9, filed February 19th 2004, the contents of which are incorporated herein by reference.

APPENDIX A

For file paths beginning with a leading '/':

/attrs	Like the unix /proc directory, stores actor attribute values for reference by content when the attribute is needed as a file reference.
<actor>	Each subdirectory of /attrs is the actor name.
<attribute>	Each attribute is accessed as a node in the actor subdirectory
<field>	If the attribute is a structure, then the field name specifies which structure member to access.
<index>	If the attribute is a vector attribute, then the index number specifies the index into the vector of the desired attribute.
<field>	If the vector attribute is a collection of structures, then the field name again specifies the structure member.

File paths without a leading '/' are treated as relative to the current trig, i.e. every trig is stored in its own folder hierarchy rooted in a single folder.

config	Common folder in every trig to store trig meta data.
channels	Common folder to store the update channel definitions.
<channel defs>	Set of files defining the collection of update channels for the trig. Each file can define one or more update channels.
start-up	Common folder to store entry points for the trig.
default	Common TrigML file to store the default entry point for the trig.
firsttime	Common TrigML file to store the TrigML for use the first time this trig is run
<trigml files>	Other named TrigML files can be used as entry points if found in the start-up folder.
constants	This folder is not passed OTA and is instead fully resolved at content compile time.
<rest of content>	trig content is organised in trig-defined format under the Trigs folder.

APPENDIX B

Trigplayer Actor	Attributes	UpdateState	
	Messages	exit	
		predial_mode	on/off
	Events	idle	

Launch Actor	Attributes		
	Messages	browser	url
		SMS	Number
			message
		Camera	
		Inbox	
		Profiles	
		missed_calls	
		dialer	number
		...	
		native_app	app_id
			url
	Events		

5

Install Actor	Attributes		
	Messages	ringtone	resource_path
		wallpaper	resource_path
	Events		

Phone Actor	Attributes	Bluetooth	
		IrDA	
		Call	
		GPRS	
		UnreadSMS	
		UnreadVoiceMail	
		UnreadMsgs	
		BatteryLevel	
		SignalStrength	
	Messages		
	Events	missed_call	
		message_arrived	
		voice_mail_arrived	

APPENDIX C

<trigml> **<layer>** id	***Listener Elements*** *common attributes* when consume
Visible Elements *common attributes* id x y w h bdcolor bgcolor hasfocus canfocus clip raise **<group>** **<grid>** rows cols rowsplit colsplit **<griddata>** repeatover	**<throw>** event target **<att>** name value valuefrom **<anim>** name duration repeat persist startvalue endvalue bounce **<load>** res target **<setvar>** **name value valuefrom**

rows cols rowsplit colsplit **<gridlist>** initrow initcol rows cols rowsplit colsplit **<image>** res frames index **<tile>** res bdt bdb bdr bdl **<text>** res font size slant weight align color fxcolor multiline **<paintif>** res isvalid **<ticker>** repeatover **<batterylevel>** res frames **<signalstrength>** res frames **<phonestatus>** res include **<include>** res **<param>** name value valuefrom	***System Events*** **entry** **focus !focus** **keypress** [key] **!keypress** [key] **longkeypress** [key] **!longkeypress** [key] **moreUpChanged** [newValue] **moreDownChanged** [newValue]

type: visible		Class of element that can have a visible representation on the display. This section
contains	contained by	

any *listener*	any *container*		describes attributes and properties common to all *visible* elements.
attributes	type	default	
id	*string*	*none*	The name or ID of this element. This identifier is used in the target attribute of <throw> and <load> elements. If the same ID is used more than once, then the last ID loaded is used.
x *(modifiable)*	*integer* left centre center right	centre	The x-coordinate of the frame of the element, relative to the top-left corner of the parent element. If one of 'left', 'centre' or 'right', the frame is suitably aligned within parent element.
y *(modifiable)*	*integer* top centre center bottom	centre	The y-coordinate of the frame of the element, relative to the top-left corner of the parent element. If one of 'top', 'centre' or 'bottom', the frame is suitably aligned within parent element.
w *(modifiable)*	*integer* *	*	The width of the frame of the element. If '*', the frame assumes the width of the parent frame, or cell, if it is in a grid.
h *(modifiable)*	*integer* *	*	The height of the frame of the element. If '*', the frame assumes the height of the parent frame, or cell, if it

			is in a grid.
bgcolour, bgcolor *(modifiable)*	*colour*	#00000000 (trans-parent)	The background fill colour of the element. If translucent alpha values are not supported, then the alpha component will round down to fully transparent.
bdcolour, bdcolor *(modifiable)*	*colour*	#00000000 (trans-parent)	The colour of the border for this element. The border is drawn 1-pixel wide and just *inside* the frame. The border can be partially or fully obscured by the child contents. If translucent alpha *is not supported*, then the alpha component is rounded up to full opacity.
clip	*boolean*	true	If true, the painting of all child contents of this element will be clipped by the frame of this element, i.e. children cannot 'spill' outside the frame. If false, the painting of all child contents will be clipped by the clipping frame of the parent element. clip=false should be used with caution as it slows down the renderer.
raise *(modifiable)*	*boolean*	false	If true, the painting of this element is painted last within its <layer>. If more than one element specifies raise=true, then they are all painted last, but in their normal

			relative order. If false, the painting of this element is in the normal order - that of painting elements in the order parsed.
hasfocus	*boolean*	false	If true, this element will be given the initial focus for the layer that it is in. If more than one element specifies hasfocus=true, then the last within each layer to do so is given the initial focus. When loading new content that contains an element with hasfocus=true, the focus is only given to this element if the new content is removing the element that previously had the focus.
canfocus	*boolean*	false	If true, this element will be given the focus when navigating with the cursor keys. If false, this element will be ignored when navigating with the cursor keys. (Note: This replaces: <att when=focus/>)

<trigml>		The root element of all TrigML documents. It does not have any visual appearance.
contains	contained by	
any element	*none*	
attributes	type default	

none

<layer>			Full screen layer. Each layer manages its own focus. The highest layer with a non-null focus element gets keypresses and events sent to _top.
contains any *visible* any *listener*	contained by *_top*		
attributes	type	default	
id	*string*	*none*	The name or ID of this element. This identifier is used in the target attribute of <throw> and *<load> elements. If the same ID* is used more than once, then the last ID loaded is used.

<group> *visible, container*			Generic container of other elements. Can be used as a plain rectangle.
contains	contained by		
any *visible* any *listener*	any *container*		
attributes	type	default	
all attributes in *type: visible*			

<grid> *visible, container*		Container element that arranges its children in a grid. <grid> is purely for layout. Use
contains	contained by	

any *visible* any *listener*	any *container*		<gridlist> *or* <griddata> for focus management. Each child is placed in its cell, with that cell forming its parent frame - i.e. children that leave w/h as '*' will be the size of their cell.
attributes	type	default	
rows	*integer*	*none*	The number of rows in the grid. Cannot be zero. If rows is supplied and cols is not, then the grid is filled column by column.
cols	*integer*	*none*	The number of columns in the grid. Cannot be zero. If cols is supplied and rows is not, then the grid is filled row by row. If both rows and cols are supplied, then the grid is also filled row by row.
rowsplit	*list of semi-colon separated integers or *s*	*	The heights of each row. If fewer values are supplied than there are rows, the last value is repeated for each extra row. All rows that have * for a rowsplit share the available space.
colsplit	*same as rowsplit*	*	The column width equivalent of rowsplit.
all attributes in *type: visible*			Note that clip applies to the whole grid, not each cell in the grid.

<gridlist> *visible, container*			Container element that arranges its children in a grid. It is also a Focus Manager in that it moves an active cell around the grid, scrolling the grid if the grid is bigger than the frame of this element. Note that both rows and rowsplit, and cols and colsplit, must be supplied to achieve a grid that is larger than the w/h of this element.
contains	contained by		
any *visible* any *listener*	any *container*		
attributes	type	default	
initrow	*integer*	*0*	The initial row of the active cell. Count from zero. See initcol below.
initcol	*integer*	*0*	The initial column of the active cell. Count from zero. The first time the gridlist gets focus, this is the cell that is in turn given focus. The hasfocus attribute overrides initrow and initcol.
all attributes in *<grid>*			
all attributes in *type: visible*			Note that clip is always true for a gridlist.

<griddata> *visible, container*		Container element that treats its single child, or single rows-worth of children, as a
contains	contained by	

any *visible* any *listener*	any *container*		template for the rest of the cells in the grid. If the special variable $$ appears in the definition of the child template, then it is replaced with the current scroll position in the set of values defined by the repeatover attribute. Only the number of children that fit in the grid are used, with the value of $$ being scrolled as focus is moved up and down the grid.
attributes	type	default	
repeatover	*resource path*	*No default. Must be supplied.*	*Specifies the set of values* to use for the $$ variable in the child elements. If the resource path is a folder, then the list of resources found in that folder are used (in numeric order) for the set of values for $$. If the resource path is a file, then the file is treated as an index file that specifies a list of values for $$.
all attributes in *<grid>*			
all attributes in *type: visible*			

<image> *visible*			Draws an image.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
res *(modifiable)*	*resource path*	*none*	The resource path of the PNG file. Image is a transparent blank if res is not suppplied.
frames	*integer*	*1*	The number of frames (side by side images) in the PNG file. The image width is therefore the real PNG width divided by the number of frames.
index *(modifiable)*	*integer*	*1*	The frame number (counting from 1) to display.
all attributes in *type: visible*			The default for w/h is to shrink to fit the supplied image. If the image is not found, then w/h default as normal. If w/h are supplied, the image is aligned to the top left corner.

<tile> *visible*			Draws a tiled image. If borders are also supplied, the image is tiled by preserving corners and edges, tiling these lengthways as necessary.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
res *(modifiable)*	*resource path*	*none*	The resource path of the PNG file. The tile is transparent

			blank if the res is not supplied.
bdt	*integer*	*0*	The thickness of the top border. If zero, the tiling has no top edge tile.
bdl	*integer*	*0*	The thickness of the left border. If zero, the tiling has no left edge tile.
bdr	*integer*	*0*	The thickness of the right border. If zero, the tiling has no right edge tile.
bdb	*integer*	*0*	The thickness of the bottom border. If zero, the tiling has no bottom edge tile.
all attributes in ***type: visible***			

<text> *visible*			Draws a text string. Text can be single or multiline, scrollable or not, editable or not. Text is drawn with device specific fonts.
contains any *listener*	contained by any *container*		
attributes	type	default	
res *(modifiable)*	*resource path*	*none*	The resource path of the text string to display (initially if editable). A transparent blank is drawn if not supplied.
font	fixed serif *sansserif* system	serif	Device specific font.
size	small medium large	small	Device specific size. Should map to 9pt, 12pt and 18pt respectively.
weight	plain bold	plain	Device specific weight for the font.
slant	plain italic	plain	Device specific weight for the font.
align	left centre center right	left	The horizontal alignment of the text string inside the frame of the text box. There is no vertical alignment control, use the y attribute to control the text box position instead.
color, colour *(modifiable)*	*colour*	#ff000000 (black)	The colour of the text. If translucent alpha is not supported, the alpha component is rounded up to full opacity.

fxcolor, fxcolour *(modifiable)*	*colour*	*#00000000*	The colour of the text effect. The default text effect is a glow background.
multiline	*boolean*	*false*	If false, the string is drawn on a single line. The width of this element will default to the length required to exactly fit the string. If true, the string will be drawn on multiple lines. The width will default to be the same as the parent element. The height will default to the height required to exactly fit the number of lines for the string.
scrollable	*boolean*	*false*	If true, the view of the string can be scrolled (horizontally for single line, vertically for multiline) when this element has the focus. Focus is released when the end or beginning of the string is reached, or if a cursor key is pressed in the non-scrolling direction.
editable	*resource path to writable resource*	*None*	If supplied, this element is an editable text box. Text editing is drawn in a device specific way, and may involve pressing select to activate text editing. The edited value of the string

			is stored in the resource path supplied by this attribute.
all attributes in *type: visible*			

<throw> *listener*			Throws an event. Events can be sent to other parts of the content tree or to an actor.
contains	contained by		
<param>	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only trigger when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
event	*event name*	*none*	The name of the event to throw. If this is an Actor event, it will automatically be sent to the relevant Actor, regardless of the specified target. Use square brackets to specify an anonymous parameter value to accompany this event. Use <param> children to specify named parameters for this event.

			If the event is the 'focus' event, then this will cause the focus to move to the target element (within the layer of the target element).
target	*element ID*	_top	The element ID of the element to send this event to. If not supplied then _top is used. If the event is an Actor event, this attribute is ignored.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<att> *listener*			Modifies an attribute of its parent *visible* when switched on. <att> is switched on by the event specified in the when attribute. It is switched off by the '!' version of the event. If several <att>s modify the same parent attribute, the last <att> that is switched on wins.
contains	contained by		
	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only switch on when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event

			when the parameter value is '_select'
name	*attribute name*	*none*	The name of the attribute in the parent *visible* to modify. The attribute must be *modifiable* as indicated in the attribute boxes in this spec.
value	*same as attribute being modified*	*none*	The new value for the named attribute of the parent *visible*. Use the @-symbol to reference the value of a named parameter of the incoming event.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<anim> *listener*			Continuously modifies an attribute of its parent *visible* while switched on. The animation is started by the event, and restarted every time the event arrives subsequently. The modification (whereever the animation has got to) is switched off when the '!' version of the event arrives.
contains	contained by		
	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only switch on when

			the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
name	*attribute name*	*none*	The name of the attribute in the parent *visible* to modify. The attribute must be *modifiable* as indicated in the attribute boxes in this spec.
startvalue	*same as attribute being modified*	*none*	The value to use at the start of the animation. If not supplied, the current value is used. The current value depends on all previous *listener* elements that modify the same attribute and the value specified by the parent *visible* itself.
endvalue	*same as attribute being modified*	*none*	The value to use at the end of the animation. This value is reached at the time specified by the duration attribute. If not supplied, the current value of the attribute is used in the same way as startvalue above.
duration	*integer number of milliseconds*	*300*	The length of time taken to animate the named attribute from startvalue to endvalue once. Note this is not the total duraction of the animation which can be

			calculated by multiplying the number of repeats by this duration.
repeat	*integer* -1 =*forever*	0	The number of times to repeat the animation after the first time through, i.e. setting it to 1 will result in the animation being played twice.
bounce	*boolean*	false	If true, the animation will play backwards on alternate repeats.
persist	*boolean*	*depends...*	If true, the animation will hold the endvalue as the modification until switched off by the '!' event. If false, the animation will revert to the startvalue at the end of the animation and hold that value until the animation is switched off. The default depends whether the event is a normal event or a '!' version of an event. If the event is normal, the default is true. If the event is a '!' event, the default is false.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<load> *listener*			Loads some new content into the supplied target element.
contains	contained by		
<param>	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only trigger when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
res	*resource path*	*none*	The resource path of the trigml file to load.
target	*element ID*	_top	The element ID to replace the children of.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<include>			Inlines the specified trigml file. The trigml in the file is treated as if it had been originally declared in place of this <include> element.
contains	contained by		
<param>	any *element*		
attributes	type	default	
res	*resource path*	*none*	The resource path of the trigml file to include.

<param>			Supplies a parameter name and value to a <load>, <include> or <throw> element.
contains	contained by		
	<load> <include> <throw>		
attributes	type	default	
name	*parameter name*	*none*	The name of the parameter. The $-symbol is used to reference the parameter when used in a <load> or <include>. The @-symbol is used to reference the parameter when used with an event.
value	*value*	*none*	The value of the parameter.
valuefrom	*resource path*	*none*	The resource path of a file to read the contents of to obtain the value of this parameter.

<setvar> *listener*			Sets a variable. The variable can only used when loading new content. If <setvar> triggers on the 'entry' event, the variable cannot be used until the next <load> tag is used.
contains	contained by		
<param>	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only trigger when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
name	*variable name*	*none*	The name of the variable.

value	*value*	*none*	The value to put in the variable. The variable can be referenced with the $-symbol in subsequent <load> actions.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<paintif> *container, visible*			Only paints its contents if the specified resource exists or the path is valid. The contents are still in the tree, and still respond to events, however, none of the contents are painted if the condition is not met. <paintif> can be used in place of group.
contains	contained by		
any *visible* any *listener*	any *element*		
attributes	type	default	
res	*resource path*	*none*	The resource path to test for the existence of.
isvalid	*resource path*	*none*	The resource path to test the validity (as a resource path) of. Note this will not actually check if the file exists, merely whether or not the path is a valid path. This is useful for testing whether $$ is in range or not.

<ticker> *visible*		Scrolls a series of items onto, then off, the frame of this element. The visible child element
contains	contained by	

any *listener* any one *visible*	any *container*		of <ticker> is used as a template for each item. Each item is scrolled on from below the element up into a centre-left-aligned position. The item is then paused before scrolling it off to the left. Use the $$ variable in the template to vary the item on each scroll past. The list is restarted at the top when the last item has been scrolled past.
attributes	type	default	
repeatover	*resource path*	*No default. Must be supplied.*	Specifies the set of values to use for the $$ variable in the child elements. If the resource path is a folder, then the list of resources found in that folder are used (in numeric order) for the set of values for $$. If the resource path is a file, then the file is treated as an index file that specifies a list of values for $$.
all attributes in *type: visible*			

<batterylevel> *visible*			Draws the battery level using the supplied image as a multi-framed image. The current value of the battery level is mapped onto the proportional frame number.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	

res *(modifiable*	*resource path*	*none*	The resource path of the PNG file that holds all the states of the battery level.
frames	*integer*	*1*	The number of frames (side by side images) in the PNG file. The image width is therefore the real PNG width divided by the number of frames. The frame that is displayed depends on the current battery level.
all attributes in *type: visible*			The default for w/h is to shrink to fit the supplied image. If the image is not found, then w/h default as normal. If w/h are supplied, the image is aligned to the top left corner.

<signalstrength> *visible*			Draws the signal strength level using the supplied image as a multi-framed image. The current value of the signal strength level is mapped onto the proportional frame number.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
res *(modifiable*	*resource path*	*none*	The resource path of the PNG file that holds all the states of the signal strength level.
frames	*integer*	*1*	The number of frames (side by side images) in the PNG file. The image width is therefore the real PNG width divided by the number of frames. The frame that is displayed depends on the current signal strength level.
			The default for w/h is to shrink

all attributes in *type: visible*			to fit the supplied image. If the image is not found, then w/h default as normal. If w/h are supplied, the image is aligned to the top left corner.

<phonestatus> *visible*			Draws a row of phone status icons. The icons are packed together, and are drawn, left to right, in the order specified in the include attribute. Use a blank image in order to reserve a space for an icon that is currently not visible.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
res *(modifiabl*	*resource path*	*none*	The root folder for the collection of icon images. For each capability specified by the include attribute, this element will look for a folder of the same name. Within that folder, this element will look for an image with a name equal to the current value of that capability.
include	*list of semi-colon separated capability names*	*none*	The names the status icons to display. Each name is a capability and should have a folder under the root folder specified by the res attribute.
all attributes in *type: visible*			